rsm_test

irw

1/29/2021

This is a test of the RSM package and its use for running an experimental design.

It is adapted from the guide by the author, Russ Lenth. https://cran.r-project.org/web/packages/rsm/vignettes/rsm.pdf https://cran.r-project.org/web/packages/rsm/rsm.pdf

```
library(rsm)
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --

## v ggplot2 3.3.3     v purrr   0.3.4
## v tibble  3.1.1     v dplyr   1.0.5
## v tidyr   1.1.3     v stringr 1.4.0
## v readr   1.4.0     v forcats 0.5.1

## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

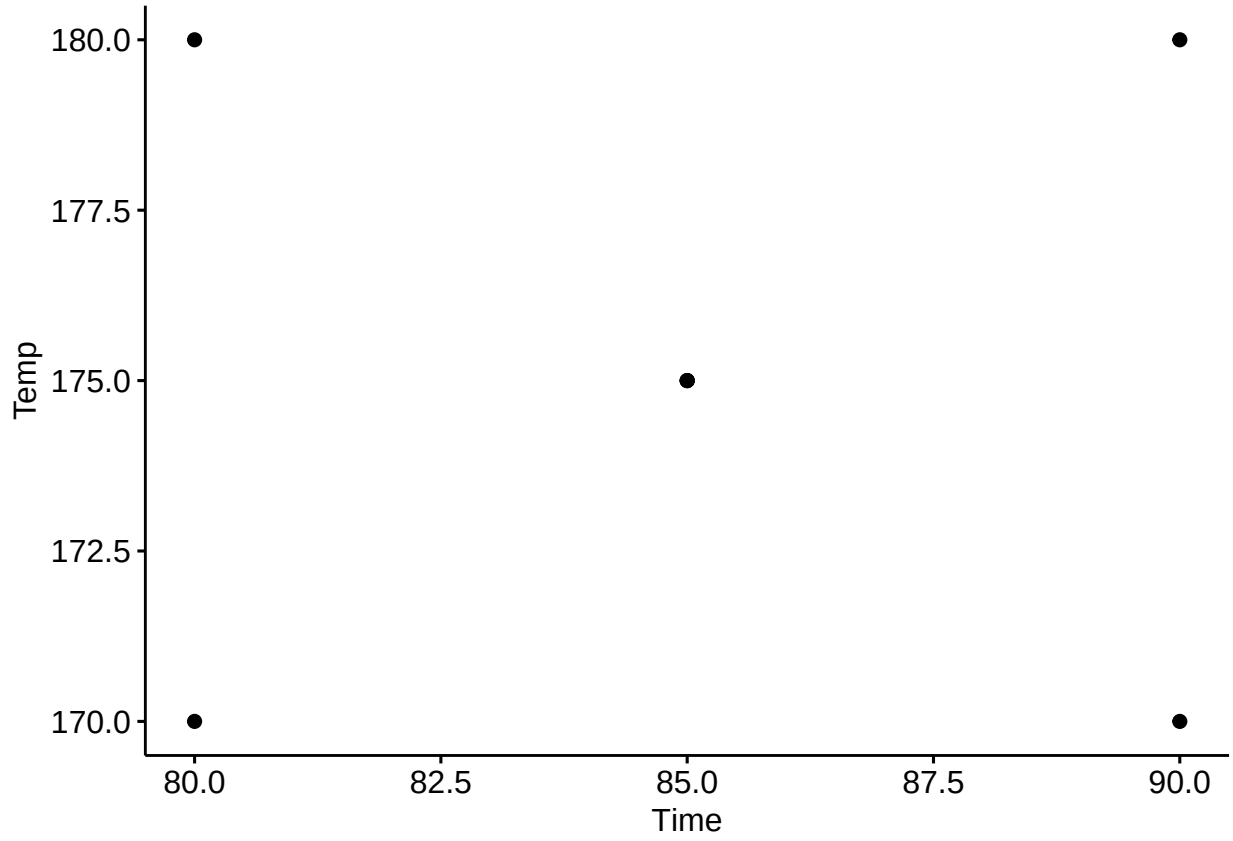
```
library(ggpubr)
```

Let's look at some of the data provided in the `rsm` package to see what a central composite design looks like. ChemReact was a dataset collected in 2 parts (blocks). First, the data in ChemReact1 was collected. Then the data from ChemReact2 was collected for further analysis.

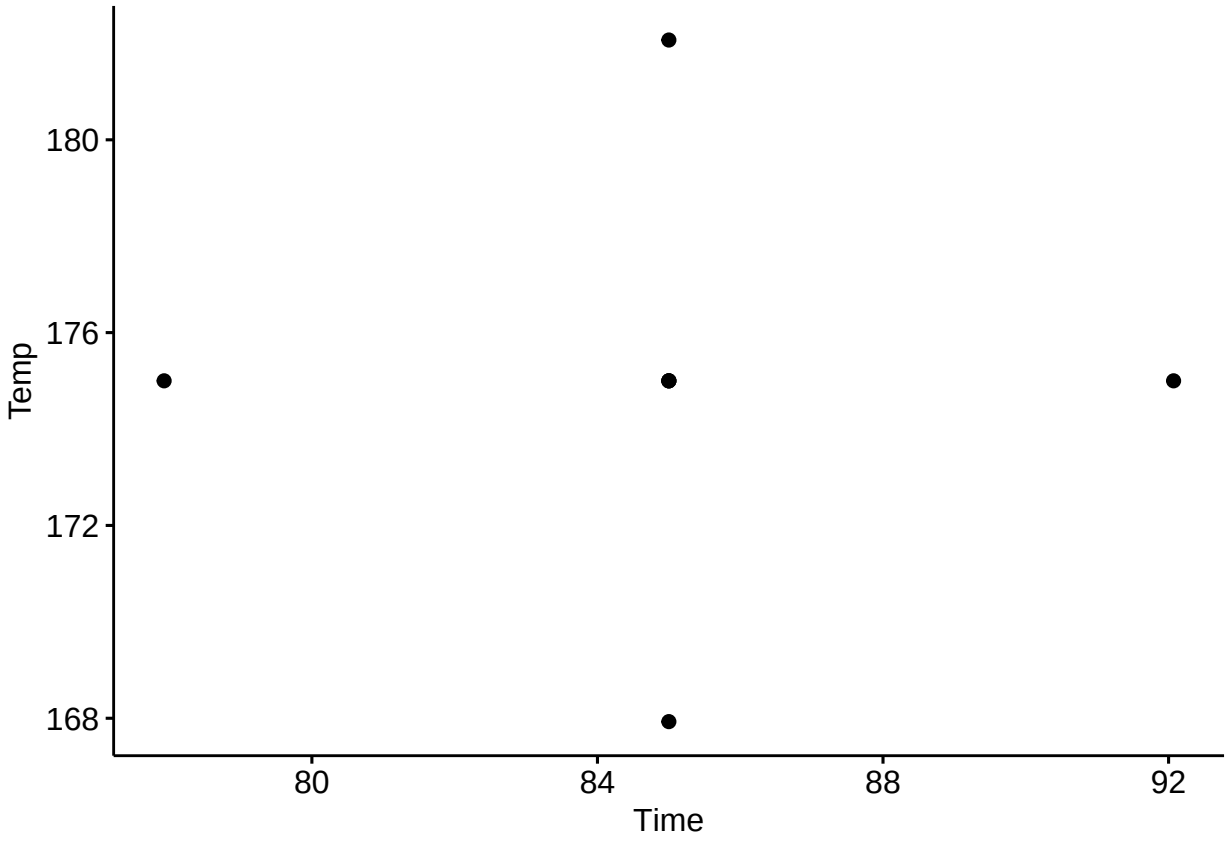
```
ChemReact
```

```
##     Time   Temp Block Yield
## 1  80.00 170.00    B1  80.5
## 2  80.00 180.00    B1  81.5
## 3  90.00 170.00    B1  82.0
## 4  90.00 180.00    B1  83.5
## 5  85.00 175.00    B1  83.9
## 6  85.00 175.00    B1  84.3
## 7  85.00 175.00    B1  84.0
## 8  85.00 175.00    B2  79.7
## 9  85.00 175.00    B2  79.8
## 10 85.00 175.00    B2  79.5
## 11 92.07 175.00    B2  78.4
## 12 77.93 175.00    B2  75.6
## 13 85.00 182.07    B2  78.5
## 14 85.00 167.93    B2  77.0
```

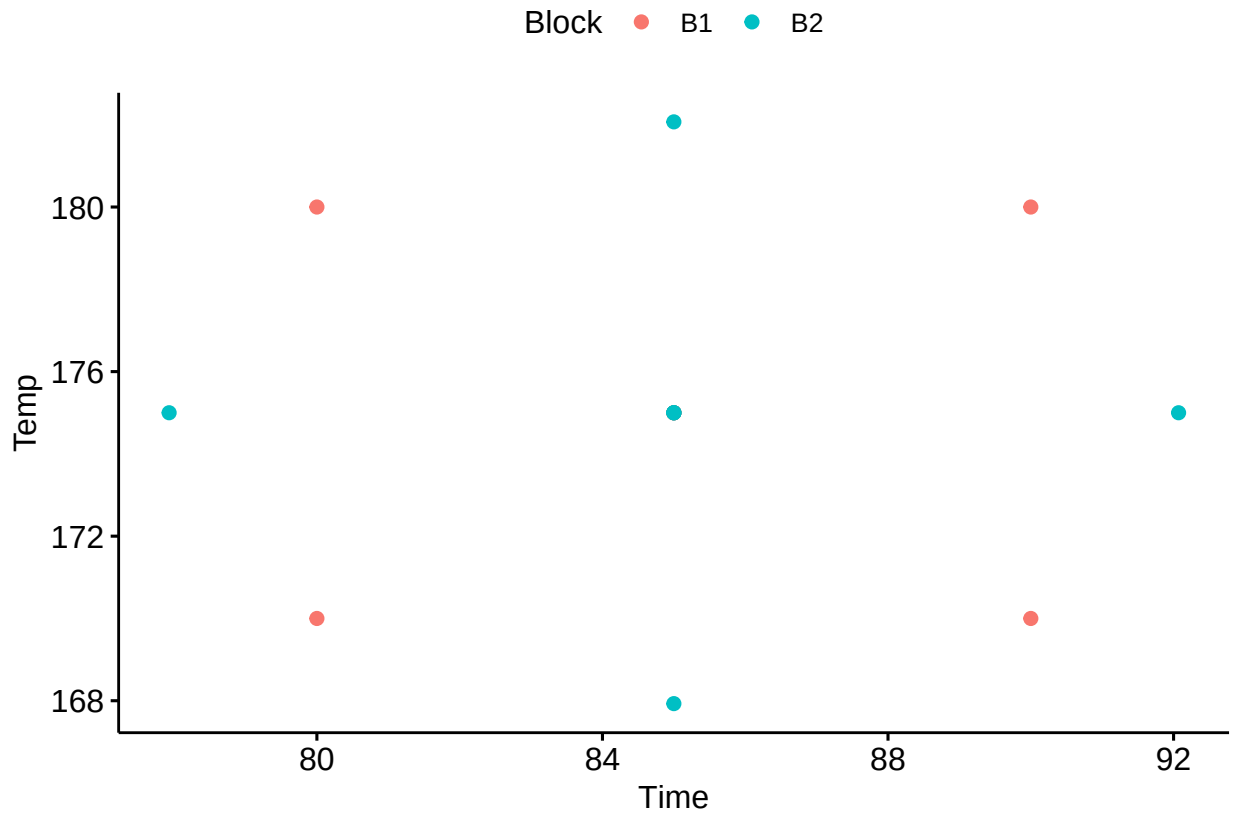
```
ggscatter(ChemReact1, "Time", "Temp")
```




```
ggscatter(ChemReact2, "Time", "Temp")
```




```
ggscatter(ChemReact, "Time", "Temp", color = "Block")
```



In order to use this data with the `rsm` functions, it has to be coded. The values of time are centered around 85 and vary by +/-5; with temperature, the values are centered about 175 +/- 5.

```
CR1 <- coded.data(ChemReact1, x1 ~ (Time - 85)/5, x2 ~ (Temp - 175)/5)
CR1
```

```
##   Time Temp Yield
## 1   80  170  80.5
## 2   80  180  81.5
## 3   90  170  82.0
## 4   90  180  83.5
## 5   85  175  83.9
## 6   85  175  84.3
## 7   85  175  84.0
##
## Data are stored in coded form using these coding formulas ...
## x1 ~ (Time - 85)/5
## x2 ~ (Temp - 175)/5
```

```
tibble(CR1)
```

```
## # A tibble: 7 x 3
##      x1    x2 Yield
##   <dbl> <dbl> <dbl>
## 1    -1    -1  80.5
```

```
## 2    -1     1  81.5
## 3     1    -1  82
## 4     1     1  83.5
## 5     0     0  83.9
## 6     0     0  84.3
## 7     0     0  84
```

The `ccd.pick()` function describes available central composite designs based on the input parameters.

```
ccd.pick(2)
```

```
##    n.c n0.c blks.c n.s n0.s bbr.c wbr.s bbr.s  N alpha.rot alpha.orth
## 1    4    1      1   4    1     1     1     1 10  1.414214   1.414214
## 2    4    2      1   4    2     1     1     1 12  1.414214   1.414214
## 3    4    3      1   4    3     1     1     1 14  1.414214   1.414214
## 4    4    4      1   4    4     1     1     1 16  1.414214   1.414214
## 5    4    5      1   4    5     1     1     1 18  1.414214   1.414214
## 6    4    6      1   4    6     1     1     1 20  1.414214   1.414214
## 7    4    7      1   4    7     1     1     1 22  1.414214   1.414214
## 8    4    8      1   4    8     1     1     1 24  1.414214   1.414214
## 9    4    9      1   4    9     1     1     1 26  1.414214   1.414214
## 10   4   10      1   4   10     1     1     1 28  1.414214   1.414214
```

The `ccd()` function shows the run order, treatments, and blocks in the central composite design selected.

```
ccd(2)
```

```
##    run.order std.order  x1.as.is  x2.as.is Block
## 1          1         7  0.000000  0.000000     1
## 2          2         5  0.000000  0.000000     1
## 3          3         3 -1.000000  1.000000     1
## 4          4         2  1.000000 -1.000000     1
## 5          5         4  1.000000  1.000000     1
## 6          6         1 -1.000000 -1.000000     1
## 7          7         6  0.000000  0.000000     1
## 8          8         8  0.000000  0.000000     1
## 9          1         5  0.000000  0.000000     2
## 10         2         1 -1.414214  0.000000     2
## 11         3         7  0.000000  0.000000     2
## 12         4         4  0.000000  1.414214     2
## 13         5         6  0.000000  0.000000     2
## 14         6         3  0.000000 -1.414214     2
## 15         7         8  0.000000  0.000000     2
## 16         8         2  1.414214  0.000000     2
##
## Data are stored in coded form using these coding formulas ...
## x1 ~ x1.as.is
## x2 ~ x2.as.is
```

The default behavior is to choose 4 replicates at the center. But from `ccd.pick()`, we can see that having one center point is an option.

```
ccd(2, n0=1)
```

```
##    run.order std.order  x1.as.is  x2.as.is Block
## 1          1         1 -1.000000 -1.000000     1
## 2          2         4  1.000000  1.000000     1
## 3          3         5  0.000000  0.000000     1
## 4          4         3 -1.000000  1.000000     1
## 5          5         2  1.000000 -1.000000     1
## 6          1         2  1.414214  0.000000     2
## 7          2         3  0.000000 -1.414214     2
## 8          3         5  0.000000  0.000000     2
## 9          4         4  0.000000  1.414214     2
## 10         5         1 -1.414214  0.000000     2
##
## Data are stored in coded form using these coding formulas ...
## x1 ~ x1.as.is
## x2 ~ x2.as.is
```

To make a response surface after defining the design and collecting the data, you have to first make a model.

```
CR1.rsm <- rsm(Yield ~ FO(x1, x2), data = CR1)
summary(CR1.rsm)
```

```
##
## Call:
## rsm(formula = Yield ~ FO(x1, x2), data = CR1)
##
##             Estimate Std. Error  t value  Pr(>|t|)
## (Intercept) 82.81429    0.54719 151.3456 1.143e-08 ***
## x1           0.87500    0.72386   1.2088    0.2933
## x2           0.62500    0.72386   0.8634    0.4366
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Multiple R-squared:  0.3555, Adjusted R-squared:  0.0333
## F-statistic: 1.103 on 2 and 4 DF,  p-value: 0.4153
##
## Analysis of Variance Table
##
## Response: Yield
##             Df Sum Sq Mean Sq F value  Pr(>F)
## FO(x1, x2)   2 4.6250  2.3125  1.1033 0.41534
## Residuals    4 8.3836  2.0959
## Lack of fit  2 8.2969  4.1485 95.7335 0.01034
## Pure error   2 0.0867  0.0433
##
## Direction of steepest ascent (at radius 1):
##        x1        x2
## 0.8137335 0.5812382
##
## Corresponding increment in original units:
##     Time     Temp
## 4.068667 2.906191
```

This model should be avoided because of the low Lack of Fit p-value (~0.01). Take a look at the model for a two way interaction. This can be done by simply updating the model.

```
CR1.rsmi <- update(CR1.rsm, . ~ . + TWI(x1, x2))
summary(CR1.rsmi)
```

```
##
## Call:
## rsm(formula = Yield ~ FO(x1, x2) + TWI(x1, x2), data = CR1)
##
##             Estimate Std. Error  t value  Pr(>|t|)
## (Intercept) 82.81429    0.62948 131.5604 9.683e-07 ***
## x1           0.87500    0.83272   1.0508    0.3705
## x2           0.62500    0.83272   0.7506    0.5074
## x1:x2        0.12500    0.83272   0.1501    0.8902
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Multiple R-squared:  0.3603, Adjusted R-squared:  -0.2793
## F-statistic: 0.5633 on 3 and 3 DF,  p-value: 0.6755
##
## Analysis of Variance Table
##
## Response: Yield
##             Df Sum Sq Mean Sq  F value   Pr(>F)
## FO(x1, x2)   2 4.6250  2.3125   0.8337 0.515302
## TWI(x1, x2)  1 0.0625  0.0625   0.0225 0.890202
## Residuals    3 8.3211  2.7737
## Lack of fit  1 8.2344  8.2344 190.0247 0.005221
## Pure error   2 0.0867  0.0433
##
## Stationary point of response surface:
## x1 x2
## -5 -7
##
## Stationary point in original units:
## Time Temp
##   60  140
##
## Eigenanalysis:
## eigen() decomposition
## $values
## [1]  0.0625 -0.0625
##
## $vectors
##         [,1]       [,2]
## x1 0.7071068 -0.7071068
## x2 0.7071068  0.7071068
```

Given the low Lack of Fit p-value, the model needs more data. Enter the second block. The `djoin` function will automatically code the variables and add the blocks.

```
CR2 <- djoin(CR1, ChemReact2)
CR2
```

```
##     Time   Temp Yield Block
## 1  80.00 170.00  80.5     1
## 2  80.00 180.00  81.5     1
## 3  90.00 170.00  82.0     1
## 4  90.00 180.00  83.5     1
## 5  85.00 175.00  83.9     1
## 6  85.00 175.00  84.3     1
## 7  85.00 175.00  84.0     1
## 8  85.00 175.00  79.7     2
## 9  85.00 175.00  79.8     2
## 10 85.00 175.00  79.5     2
## 11 92.07 175.00  78.4     2
## 12 77.93 175.00  75.6     2
## 13 85.00 182.07  78.5     2
## 14 85.00 167.93  77.0     2
##
## Data are stored in coded form using these coding formulas ...
## x1 ~ (Time - 85)/5
## x2 ~ (Temp - 175)/5
```

Now there is enough information to fit a second order model, but the blocks should be accounted for in the new model.

```
CR2.rsm <- rsm(Yield ~ Block + SO(x1, x2), data = CR2)
summary(CR2.rsm)
```

```
##
## Call:
## rsm(formula = Yield ~ Block + SO(x1, x2), data = CR2)
##
##              Estimate Std. Error  t value  Pr(>|t|)
## (Intercept) 84.095427   0.079631 1056.067 < 2.2e-16 ***
## Block2      -4.457530   0.087226  -51.103 2.877e-10 ***
## x1           0.932541   0.057699   16.162 8.444e-07 ***
## x2           0.577712   0.057699   10.012 2.122e-05 ***
## x1:x2        0.125000   0.081592    1.532    0.1694
## x1^2        -1.308555   0.060064  -21.786 1.083e-07 ***
## x2^2        -0.933442   0.060064  -15.541 1.104e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Multiple R-squared:  0.9981, Adjusted R-squared:  0.9964
## F-statistic: 607.2 on 6 and 7 DF,  p-value: 3.811e-09
##
## Analysis of Variance Table
##
## Response: Yield
##             Df Sum Sq Mean Sq   F value    Pr(>F)
## Block        1 69.531  69.531 2611.0950 2.879e-10
```

```
## FO(x1, x2)   2  9.626   4.813  180.7341 9.450e-07
## TWI(x1, x2)  1  0.063   0.063    2.3470    0.1694
## PQ(x1, x2)   2 17.791   8.896  334.0539 1.135e-07
## Residuals    7  0.186   0.027
## Lack of fit  3  0.053   0.018    0.5307    0.6851
## Pure error   4  0.133   0.033
##
## Stationary point of response surface:
##        x1        x2
## 0.3722954 0.3343802
##
## Stationary point in original units:
##      Time      Temp
##  86.86148 176.67190
##
## Eigenanalysis:
## eigen() decomposition
## $values
## [1] -0.9233027 -1.3186949
##
## $vectors
##          [,1]       [,2]
## x1 -0.1601375 -0.9870947
## x2 -0.9870947  0.1601375
```